Computer Science Department

TECHNICAL REPORT

INPUT SENSITIVE, OPTIMAL PARALLEL RANDOMIZED ALGORITHMS FOR ADDITION AND IDENTIFICATION

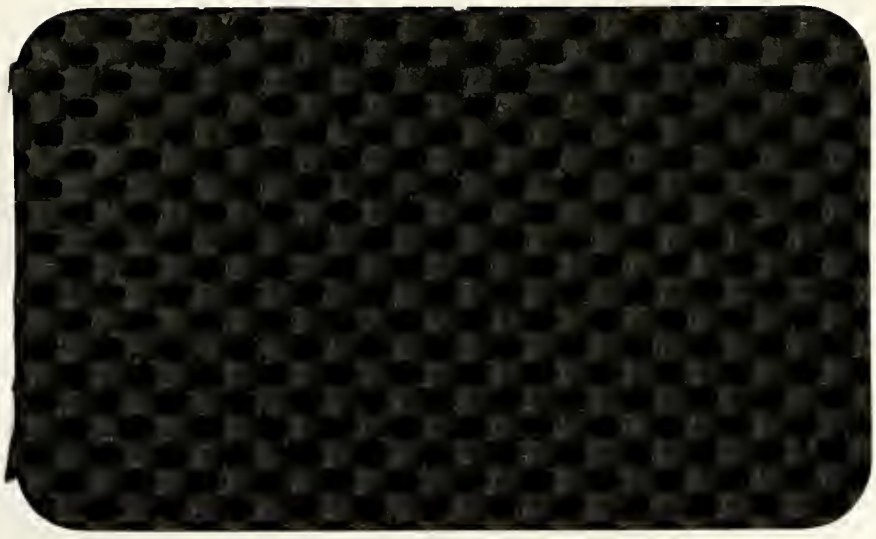
By

Paul G. Spirakis[1]
July 1985

Technical Report #182

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



INPUT SENSITIVE, OPTIMAL PARALLEL RANDOMIZED ALGORITHMS FOR ADDITION AND IDENTIFICATION

By

Paul G. Spirakis[1]
July 1985

Technical Report #182

[1]Courant Institute, New York and Computer Technology Institute, Greece

This work was supported in part by NSF grant MCS 83-00630.

Abstract

Although many sophisticated parallel algorithms now exist, it is not at all clear if any of them is sensitive to properties of the input which can be determined only at run-time. For example, in the case of parallel addition in shared memory models, we intuitively understand that we should not add those inputs whose value is zero. A technique which exploits this idea, may beat the general lower bound for addition if the count of nonzero operants is much smaller than the numbers to be added. In this paper, we device such algorithms for two fundamental problems of parallel computation. Our model of computation is the CRCW PRAM. We first provide a randomized algorithm for parallel addition which never errs and computes the result in O (logm) expected parallel time, where m is the count of nonzero entries among the n numbersto be added. This algorithm uses O (m) shared space. We then use this result to solve the following problem of processor identification : n processors are given, each keeping either a 0 or an 1. We want each processor at the end, to know which are the processors with the 1's. Our solution is randomized and sensitive to the number m of the 1's. It takes

O (min $\{$ m, n logm/logn$\}$) expected parallel time and only O (m) shared memory, capable of holding only O(n) size numbers. Combinatorial techniques of Erdos and Renyi were helpful to a part of this second result.

All our techniques enjoy the following properties : (1) They never produce an erroneous answer (2) if T is the actual parallel time and E (T) its expected value, then Prob $\{T > k.E\ (T)\} \leq n^{-c}$ where k is arbitrary and $c > 1$ is linear on k and can be specified by the algorithm implementer.

(3) m is initially unknown to our algorithms. They produce an accurate estimate of it.

1. Introduction

Recently there has been much interest in fast parallel algorithms that employ randomization. Although many sophisticated such algorithms now exist (see e.g. the proceedings the 16th STOC Conference), it is not at all clear if any of them are sensitive to properties of the input which can be determined only at runtime. For example, in the case of parallel addition (or multiplication) in shared memory models, we understand intuitively that we should not add(multiply)those inputs whose value is zero (one). Even if we manage to quickly estimate the number of nonzero inputs, we still have to organize them in an appropriate manner (e.g. pack them in shared memory), in order to perform the addition. Our goal here is to device such algorithms (by use of randomization) which are sensitive to such dynamic properties of the input and hence beat the known lower bounds (which hold for the general case).

We use the synchronous Concurrent Read - Concurrent Write (CRCW) model of parallel computation (called WRAM) (see e.g. [SV, 80], [G, 68]). This model assumes the presence of a (potentially) unlimited number of processors with (potentially) unlimited local memory in each processor. We assume our processors capable of doing independent probabilistic choices on a fixed input (This was first used by [R, 82 a,b] and [V, 83]). WRAM is like the PRAM of [W, 79] and [FW, 78] in the sence that different processors can read the same memory location at the same time. However, in the case of a simultaneous write attempt, exactly one processor succeeds in the WRAM model. We make no assumption of which one succeeds but we assume that the failed ones are notified. This can be easily implemented by having processors read the result of the "write".

We first consider the fundamental problem of parallel addition of n numbers. Our technique first provides a probabilistic estimate of the count (m) of the non-zero inputs, and then uses a probabilistic method to lay them out in shared memory and add them. The whole algorithm takes $O(\log m)$ expected parallel time, uses $O(m)$ shared space and involves only m processors. To our knowledge, deterministic WRAM algorithms for addition have to take $O(\log n)$ parallel time when at most n processors are used and n numbers are to be added.

We then examine the related problem of processor iden-

tification : n processors of a WRAM are given, each processor must find out which are the processors with the 1's. We assume that each shared memory location can fit a number of at most O (n) size. We first use a nice technique of Erdos and Renyi (ER, 63), to provide an O (n /logn) parallel time solution to the problem, if counting of O (n) units had unit cost. We then use our first result (about addition) to provide an O (min $\{m, n \log m/\log n\}$) expected parallel time algorithm for the WRAM which uses only O (m) shared space. We also give a matching lower bound for the parallel time.

All our results satisfy the following : If T is the actual parallel time of our algorithm and E(T) is its expected value, then Prob $\{T > k.\ E(T)\} \leq n^{-c}$ where $k > 1$ is any constant and $c > 1$ grows linearly with k and can be controlled by the algorithm designer.

2. The case of parallel addition

2.1. The Algorithm

Let the array M represent the shared memory. Let $\alpha \geq$ be a positive integer constant. Let each processor Pi be equipped with a local variable, $TIME_i$, intended to keep the current parallel step. Initially, each processor P_i $(1 \leq i \leq n)$ holds locally a number x_i. The goal is to compute the sum of the x_i's. Let m be the number of the nonzero x_i's. We give the algorithm in two parts : Procedure ADDITION (m') actually performs the addition, assuming an estimate $m' = cm + d$, (c, d > 1 constants) known. Function ESTIMATION produces such an estimate. So, the whole algorithm has the following high level description :

```
begin
  m' ← ESTIMATION
  ADDITION (m')
end
```

We provide the description of ESTIMATION first. In ESTIMATION, each Pi with $x_i \neq 0$ produces k estimates of m (k is a constant) through a probabilistic technique, and then does a variance-reduction process to get the final estimate. The actual

value of k is determined in the analysis.

```
Function ESTIMATION

  procedure  PRODUCE-AN-ESTIMATE

 begin

  stage 1  (Initialization)

    Processor P1 initializes a special shared
    memory location (CLOCK) to zero. Them, each Pi
    executes TIMEi ← o.

  stage 2 (Estimate)
   Processor Pi

    if  xi ≠ o  then
      begin

  (1)  Flip a fair coin (two-sided)
  (2)  If the autcome is 'tail'then
       begin
  (2a)  TIMEi : ← TIMEi +1
  (2b)  CLOCK ← TIMEi
  (2c)  go to  (1)
  end  end
comment :This is done by processors which
        flipped a 'head'

(3)  If  xi ≠ o   then

       begin

       (4)  read CLOCK into a local variable R1
       (5)  wait for 5 steps
       (6)  read CLOCK into a local variable R2
       (7)  If R1 ≠ R2  then go to  (4)
          end

 comment   At this point, every Pi with xi ≠ o
           has flipped a 'head'
```

(8) Each Pi with $x_i \neq o$ reads CLOCK and makes its value to be the current estimate.

end (of procedure PRODUCE-AN-ESTIMATE)

begin (main part of ESTIMATION)

Each Pi with $x_i \neq o$ runs procedure PRODUCE-AN-ESTIMATE k times and produces estimates $E_1, E_2, \ldots, E_k$. Then all compute

(1) $E \leftarrow (\log 2) \dfrac{E_1 + \ldots + E_k}{k}$

(2) $m' \leftarrow \exp(2).\exp(E) + d$

where $d \geq 1$ is a constant.

m' is the value returned by ESTIMATION.
We assume that it is written to a special shared memory location, so that it is available to all processors.

end

We now provide a description of procedure ADDITION (m').
It has 3 stages :

PROCEDURE ADDITION (m')

Stage 1 (Initialization)

In one parallel step, processors initialize a.m'+2 shared memory locations to zero, by executing :"Processor P_j writes a zero to M (j), if $j \leq am' + 2$. "Then, they all execute $TIME_j \leftarrow 0$

Stage 2 (Memory Marking)

Processor P_j

IF $x_j \neq 0$ then

BEGIN

(1) Select y equiprobably at random from $\{1,2,\ldots,am'\}$

(2) $TIME_j \leftarrow TIME_j + 1$

(3) Read M (y); $TIME_j \leftarrow TIME_j + 1$

(4) If M(y) = 0 then write x_j into M (y).
Also, $TIME_j \leftarrow TIME_j + 1$

(5) If the "write" failed then
BEGIN
(5a) write $TIME_j$ into M (am'+1)
(5b) go to (1)
END
END

Comment : This part is executes by P_j with x_j =0 and by "successful" P_j with $x_j \neq 0$.

(6) Read M (am'+1) into a local variable R1

(7) Wait for 8 steps

(8) Read M (am'+1) into a local variable R2.

(9) If R1 ≠ R2 then go to (6)

Comment: R1 = R2 means all processors with $x_j \neq 0$ succeeded into writing x_j in a shared memory location, different for each processor, among M (1),..., M (am'). (If a processor was failing, the value of M(am'+1) would change).

Stage 3 (Addition)

(Processor P_j is assigned to location M(j), $1 \leq j \leq am'$)

From this point on, processors P_j (where $1 \leq j \leq am'$) perform a standard parallel addition of the numbers M(1),..., M(am'). In the ith parallel step of the addition, processor P_j adds M (j) and $M(j+2^i)$ into M(j), for $j=k.2^i+1$, $k = 0,1,2,\ldots$ $am'/2^i$. (See e.g. (K, 82) or (FW, 78) on how to do the parallel addition of am' numbers by am' processors in O (m') space and O (log m') parallel time).

2.2. Analysis of the Algorithm

Lemma 1 At the end of each execution of procedure PRODUCE-AN-ESTIMATE, the variable CLOCK is a random variable, whose mean and variance satisfy :

(1) $E(CLOCK) \cdot \log 2 \geq \log m + 0.5$

(2) $(E(CLOCK) - 1) \cdot \log 2 \leq \log m + 0.5$

(3) var (CLOCK) $\leq$ 4

Sketch of Proof (Details in full paper)

CLOCK is the maximum of m independent geometric random variables $X_1, \ldots, X_m$ (the number of coin flips until a 'head' of the Pi's with $x_i \neq 0$) with density Prob $\{ Xi = j \}$ = $(1/2)^j$ $j \geq 1$. The rest is a relatively easy calculation, since Prob $\{ \text{CLOCK} \leq j \}$ = (Prob $\{ X_1 \leq j \}$)^m

Lemma 2 Given any $\delta > 0$, if we choose $k \geq 4/\delta$ then, with probability at least $1-\delta$, we have

(1) $|E - \log m| \leq 2$

and (2) The total running time of ESTIMATION is

$$0 \left(\frac{4}{\delta} . \log m \right).$$

Proof sketch (Complete Proof in full paper)

From Chebyshev inequality and Lemma1

we get Prob $\{ |E - \log m| \leq 1.2 \} \geq 1 - 4/\delta$

Also note that the running time of ESTIMATION is $0 (k.E) = 0 (E_1 + \ldots + E_k)$

Corollary 1 Given any $\delta > 0$, if we choose $k \geq 4/\delta$ then, with probability at least $1-\delta$ we have

$$m \leq m' \leq m. \exp(4)$$

Proof It follows immediately, by Lemma 2.

In the following we assume $k \geq 4/\delta$ for a fixed small δ.

Lemma 3 Conditioned on the event

$\mathcal{E} = \{ m \leq m' \leq m. \exp(4) \}$, the time of stage 2 of procedure ADDITION (m') has an expected value of O(logm). Furthermore, the (conditional on $\mathcal{E}$) probability that the time of stage 2 exceeds β.logm is $\leq m^{-\beta \log a+1}$ (and can be made arbitrarily sma

Proof sketch (See full paper for details)

It is easy to see that every time a processor P_j attempts to write its x_j, and if $g \leq m$ shared memory locations are already "occupied", the competitors of P_j are m-g-1. Even if all of them manage to select different memory locations which were not occupied previously, the maximum number of locations that P_j must "avoid" is g + m-g-1 =m-1.
So, P_j will succeed with probability

$$\text{at least } \frac{am' - (m-1)}{am'} \geq \frac{am' - (m'-1)}{am'} \geq \frac{a-1}{a}$$

in each trial (and this holds for every P_j).

A generalization of Lemma 1 about the maximum of m geometrics with success probability $\geq 1 - 1/a$ implies that the average number of parallel steps required for all m processors to succeed is O(logm') =O (logm).

The probability that there exists a processor which continues failing for at least logm rounds is

$$\leq m. \left(\frac{1}{a}\right)^{\beta \log m} \leq m^{-\beta \log a + 1}$$

□

It is easy to see that the algorithm uses O(m') shared memory, O(m') processors, and performs the addition correctly, because, at the end of stage 2 of ADDITION(m') the m nonzero x_i's are placed one in each of m shared memory locations, and these locations are among M (1),..., M (am'). The rest of these locations contain zeros. So, we have :

Lemma 4 Given any $\delta \in (0,1)$, we can choose β 0 such that with probability at least

1- max (δ , $m^{-\beta \log a + 1}$), our algorithm performs the parallel addition in O (logm) time, uses O(m) shared space and O(m) processors. Our algorithm never errs. With diminishingly small probability, it may choose a bad estimate m' of m and hence it may never exit the loop (6)-(9) of stage 2 of ADDITION(m').

3. The processor identification problem.

3.1. An $O\left(\frac{n}{\log n}\right)$ parallel time algorithm which assumes unit-cost addition

The processor identification problem assumes that n processors are given, each keeping either a 0 or an 1. The problem is for each processor to find out which are the processors with the 1's. We first solve this problem for the so-called strong W-RAM (SW-RAM) model. This model has the property that simultaneous writes on the same memory location succeed only if they write the same value, and, if that is the case, their sum is recorded. We also assume that each shared memory location can hold only up to O(n) size-numbers. Let us imagine that all the processors are equipped with the same list $L = l_1, l_2, \ldots l_s$ of "testing" sequences, where each l_i is an n-bit sequence of 0's and 1's. Let us also assume that L is independent of the particular assignment of 0's and 1's to processors. In the following, let v be a fixed memory location and let e_i be the value of processor P_i. Processors execute the following sequence of steps, s^1 times :

Round i $(1 \leq i \leq s)$

(1) P_1 erases v's contents by writing a 0

(2) Each P_j $(1 \leq j \leq n)$ looks at the j^{th} position of l_i, If $[l_i](j) = 1$ and $e_j = 1$ then P_j writes e_j to location v.

(3) Each P_j reads v's contents.

At the end of the s rounds, each processor has, for each testing seqeunce, the number of places in which an 1 stands both in the testing seqeunce and in the sequence $e_1 e_2 \ldots e_n$ to be guessed. If L allows each processor to find $e_1 \ldots e_n$ after the s rounds, we call L an s-algorithm for the processor identification problem (We allow unrestricted local memory per processor).

An obvious L (which would take O(n) parallel time) is that consisting of n li's with $l_i(j) = 0$ for $i \neq j$ and $l_i(i) = 1$ for all i.

Erdos and Renyi (ER, 63) considered a very closely related problem, the "coin-weight" problem. Using their techniques, we show that the s needed is $\Theta(n/\log n)$ and that L can be easily constructed.

Let us view L as an sxn matrix of 0's and 1's.

Lemma 5 (see also (ER, 63)). A matrix L, sxn of 0's and 1's is an s-algorithm for the processor identification problem iff :For each pair c, c' of subsets of the set C

of columns of L, such that $c \neq c'$, if we form the row-sums of the submatrices L (c) and L (c') (consisting of the selected columns) and denote by $\vec{V}_c$ and $\vec{V}_{c'}$, the column-vectors consisting of these row-sums, then $\vec{V}_c \neq \vec{V}_{c'}$.

Proof sketch After m rounds, each processor has a row-sum vector, v , of L. This corresponds to just one subset c of the set of columns of L. This subset determines the processors with the 1's, because c is exactly the subset of processors with a value equal to 1.

Here, the techniques of (ER, 63) can be used to prove :

Lemma 6 A matrix L , sxn, with $s = an/(\log n)$, $a \geq 6$, chosen so that the sn entries are independent random variables each taking on the values 0, 1 with probability 1/2 , is an s-algorithm, with probability tending to 1 as $n \rightarrow +\infty$.

Proof sketch Let p = prob {L is an s-algorithm}

Let q = 1-p

Let E (c_1, c_2), where c_1, c_2 are subsets of the set of columns C of L, denote the event that $\vec{v}_{c_1} = \vec{v}_{c_2}$ (where $\vec{v}_{ci}$ is the row-sum vector of L (c_i) If c_1, c_2 are not disjoint, then if $d_1 = c_1 - c_1 \cap c_2$ and $d_2 = c_2 - c_1 \cap c_2$, we have $\vec{v}_{d_1} = \vec{v}_{d_2}$. Hence, if L is not an s-algorithm, there exist disjoint subsets of the set of columns, such that $v_{d_1} = v_{d_2}$. So

$$q \leq \Sigma \quad \text{prob} \{E(d_1, d_2)\}$$

where d_1, d_2 disjoint subsets of C.

One can then get, by some conbinatorics, that

$$q \leq 2^{n(\log 3 - a/2) + o(n)}, \text{ for } s = \frac{an}{\log n}$$

So, if we choose $a > \log_2 9 + 2$ then

we get $q \leq 2^{-n}$ and $\lim_{n \rightarrow \infty} q = 0$

Corollary There exists an s-algorithm, for

$$s = \frac{an}{\log n} \quad , \quad a \geq 6.$$

Proof

Since, from Lemma 6, $q < 2^{-n}$, we get $p > 1-2^{-n} > 0$. (In fact the vast majority of random o-1 sxn matrices are s-algorithms).

3.2. An $O\left(\min\left\{m, n\frac{\log m}{\log n}\right\}\right)$ algorithm for the WRAM.

In the following, let m be the number of ones among the $e_1, e_2, \ldots, e_n$.

(a) An O (m) parallel-time algorithm for identification.

THE MARKING ALGORITHM

(Stage 1) The WRAM runs the algorithm for parallel addition once (as explained in Section 2) for the values e_j of the processors. At the end of this process (which takes O (logm) time with high probability), each processor knows the number of ones among $e_1, e_2, \ldots, e_n$.

Stage 2)

The WRAM runs the stages 1 (Initialization) and 2 (memory marking) of the procedure ADDITION (m), with the following modification: Each time a processor marks a memory location, it writes its id instead of its value. At the end of this stage, the m id's of the processor with nonzero values have been placed "contiguously" in M(1), M(2),..., M(am).

(Stage 3) Each processor reads the memory locations M(1),..., M(am) in sequence.

Lemma 7 The marking algorithm solves the identification problem in the WRAM, in O(m), parallel time, with arbitrarily high probability.

Proof sketch

By Lemma 4 of Section 2, the first stage of the algorithm takes $O(\log m)$ time with probability at least $1-\max(\delta, m^{-\beta \log a+1})$ where δ and β can be selected by the implementer. The second stage of the algorithm takes $O(\log m)$ time with probability at least $1-m^{-\beta \log a+1}$, by Lemma 3 of Section 2. The last stage of our algorithm takes am time. Our algorithm never reports an erroneous answer. However, with diminishingly small probability, it may never terminate.

(β) An $O\left(n \frac{\log m}{\log n}\right)$ expected parallel time algorithm for the WRAM.'

The WRAM here will simulate the SW-RAM of Section 3.1., as follows :

(stage 1) The WRAM runs the algorithm for parallel addition once, for the values e_j of the processors. At the end of this process, each processor knows the number of ones (m) among the e_i's.

(Stage 2) The WRAM runs the s-algorithm, by simulating step 2 of each round, with the procedure ADDITION (m) (described in Section 2).

Lemma 8 The simulation algorithm described above, runs in $O\left(n \frac{\log m}{\log n}\right)$ expected parallel time.

Proof sketch Stage 1 runs in $O(\log m)$ expected time, by Lemma 4 of Section 2. Each round of the s- algorithm runs also in $O(\log m)$ expected time, by Lemma 3 of Section 2.

In the full paper, we also prove that

Lemma 9 If $m = o(n)$, then our simulation algorithm runs in $O(\log m)$ parallel time, with probability at least $1-m^{\frac{-\log n}{\log m}}$

(c) The conbination of the two techniques. We can have the WRAM running both algorithms (a) and (b) interleaved (one parallel step of (a), and then one parallel step of (b). When one of the two techniques terminates, the processors will stop.

4. Remarks and Lower bounds

Lemma 10 No s-algorithm can have $s < \frac{n}{\log(n+1)}$

Proof sketch Each processor needs at least $\log (2^n)_n = n$ "pieces of information" to distinguish between the 2^n possible assignments of 0's and 1's to processors. On the other hand, if k processors attempt an addition (in step 2 of each round), the amount of information obtained cannot exceed $\log (k+1) \leq \log (n+1)$ because the number of 1's among them are 0,1 or,..., or k. So, s rounds can give at most $s \cdot \log(n+1)$ pieces of information to each processor.

□

Remark

Once the processors have an s-algorithm L, then can construct a table of the possible row-sum vectors $\vec{V}_c$ and their corresponding subset c of L. Then, given any instance of the identification problem, then need O(s) rounds to find $\vec{V}_c$ and one (indexed) table access to find c and solve the problem. Another piece of the preprocessing work is the construction of L itself. It seems to us that the n processors of the WRAM will need $\Theta (n^2/\log n)$ time to agree to a common random L. Clearly, our algorithm of Section 3.1 and of Section 3.2 -(b), becomes practical in dynamic environments, where the values of the n processors change. We pose as a general open problem the construction of input-sensitive parallel algorithms for other problems (so that the "general" lower bounds are beaten). A possible candidate is graph connectivity for special types of graphs.

Acknowledgments

The author wishes to thank C.H. Papadimitriou, D. Shasha and Z. Kedem for helpful comments in previous versions of this work.

REFERENCES

(CLC, 83)
Chin, F., J. Lam and I. Chen, "Oprtimal Parallel Algorithms for the Connected Components Problem,"CACM83.

(C, 80)
Cook, S., "Towards a Complexity Theory of Synchronous Parallel Computations", Specker Symp. on Logic and Algorithms, Zurich, Feb.5-11, 1980.

(DNS, 81)
Dekel, E., D. Nassimi and S. Sahni, "Parallel Matrix and Graph Algorithms, "SIAM J. Comp. 10 (4) 1981.

(ER, 60)
Erdos, P. and A. Renyi, "On the Evolution of Random Graphs," The Art of Counting, J. Spencer Editor, MIT Press, 1973.

(ER, 63)
Erdos, P. and A. Renyi, "On two problems of information theory"Mayar Tud. Akad. Mat. Kut. Int. Kozl. 8 (1963); also in The Art of Counting, J. Spencer, Editor, MIT Press, 1973.

(G, 78)
Goldschlager, L., "A Unified Approach to Models of Synchronous Parallel Machines", Proc. 11 th sub STOC, May 1978

(G, 77)
Goldshlager, L., "Synchronous Parallel Computation", Ph. D. thesis, Univ. of Toronto, C.S Dept., 1977.

(GLR, 80)
Gottlieb, A., B Lubachevsky and L. Rudolph, "Basic Techniques for the efficient coordination of very large numbers of cooperating sequential processors, "Courant Inst. TR No. 028, Dec. 1980.

(HCS, 79)
Hirschberg, D., A. Chandra, D. Sarwate, "Computing Connected Components on Parallel Computers, "CACM 22(8) Aug.1979.

(K, 82)
Kucera L., "Parallel Computation and Conflicts in Memory Access", Info. Processing Letters Vol. 14, April 1982.

(MV, 83)
Melhorn, K., and U. Vishkin, "Randomized and deterministic simulation of PRAMs by parallel machines with restricted

granularity of parallel memories," 9th Workshop on Graph Theoretic Concepts in Computer Science, Univ. Usnabruck, June 1983.

(R, 82)
Reif, J., "Symmetric Complementation," 14th STOC, San Francisco, CA, May 1982.

(R, 82b)
Reif, J., "On the Power of Probabilistic Choice in synchronous Parallel Computations", 9th ICALP, Aarchus, Denmark, July 1982.

(Ru, 79)
Ruzzo, W., "On Uniform Circuit Complexity", Proc. 20th FOCS, Oct. 1979.

(SJ, 81)
Savage, C. and J. Ja'ja', "Fast, Efficient Parallel Algorithms for Some Graph Problems", SIAM J. Comp. 10 (4), Nov. 1981.

(SV, 80)
Shiloach, Y. and U. Vishkin, "Finding the Maximum Merging and Sorting in a Parallel Computation Model", Tech. Rep. Technion Israel, Comp. Sci., March 1980.

(SV, 82)
Shiloah, Y. and U. Vishkin. "An O (logn) Parallel Connectivity Algorithm", J. of Algorithms, 1982.

(S, 80)
Schwartz, J.T., "Ultracomputers", ACM TOPLAS 1980, pp. 484-521.

(UW, 84)
Upfal, E. and A. Wigderson, "How to share memory in a distributed system", 25th FOCS, Proceedings, October 1984.

(V. 83a)
Vishkin, U., "A parallel-design, distributed-implementation general purpose parallel computer", to appear, J. TCS.

(V, 83b)
Vishkin, U., "Randomized speeds-ups in parallel computation 16th STOC, April 1984, Proceedings.

(U. 84)
Upfal, E., "A probabilistic relation between desirable and feasible models of parallel computation", 16th ACM STOC 1984, Proceedings.

(W, 79)
Wyllie, J., "The Complexity of Parallel Computation", Ph. D. Thesis, Cornell University, 1979.

NYU COMPSCI TR-182 C.1
Spirakis, Paul G
Input sensitive, optimal
parallel randomized
algorithms...

DATE DUE	BORROWER'S NAME